Reports and Technical Reports | All Technical Reports Collection

1976-03-15

Design manual for the vector general graphics display unit

Thorpe, Lloyd Allen

Monterey, California. Naval Postgraduate School

NPS-72Rr76032

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DESIGN MANUAL FOR THE
VECTOR GENERAL GRAPHICS DISPLAY UNIT

by

L. A. Thorpe

G. M. Raetz

March 1976

Approved for public release; distribution unlimited.

Prepared for: Naval Electronics Systems Command (ELEX 320)
Washington, D. C.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Isham Linder   Jack R. Borsting
Superintendent   Provost

This report was prepared by:

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS-72Rr76032	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN MANUAL FOR THE VECTOR GENERAL GRAPHICS DISPLAY UNIT		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) L. A. Thorpe G. M. Raetz		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N0003975WR 59051 PDM 000320
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Electronics Systems Command (ELEX 320) Washington, D. C. 20360		12. REPORT DATE 15 March 1976
		13. NUMBER OF PAGES 78
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) Computer Science Group (Code 72) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Real time
Interactive graphics
Graphics

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the program description of an inter-
active graphics package interfacing the Vector General Graphics
Display Unit and a Digital Equipment Corporation PDP-11/50
computer. The program was written in the C-programming language
and designed to be used in the multiprogramming environment
of the UNIX Timesharing operating system. Included is a des-
cription of the Vector General, operating system modifications,
device driver, and interface routines.

DD FORM
1 JAN 73 1473   EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

ABSTRACT

This report describes the program description of an interactive graphics package interfacing the Vector General Graphics Display Unit and a Digital Equipment Corporation PDP-11/50 computer. The program was written in the C-programming language and designed to be used in the multiprogramming environment of the UNIX Timesharing operating system. Included is a description of the Vector General, operating system modifications, device driver, and interface routines.

TABLE OF CONTENTS

2

3

I. INTRODUCTION

This manual is the program description for an interactive graphics package interfacing a Vector General Graphics Display System and a PDP-11/50 user. The Vector General Graphics Display System (vector general) is an interactive graphics cathode ray tube (CRT) display that is connected to the PDP-11/50 computer via a modified DR11-B interface. The display interacts with an on-line user by displaying pictorial information on the surface of the cathode ray tube and by accepting inputs from external control devices. The inputs are requested and processed by computer programs that alter and maintain the output picture being presented to the user.

This manual assumes that the reader is familiar with the C-programming language and the UNIX operating system. The terminology used herein without explanation refers to the features and registers in the display unit. A more detailed description can be found in the Vector General Graphics Display Unit Reference Manual (VG 101056). A user's manual is published separately.

The software design can be divided into five major categories:

1. Modifications to UNIX

2. System Routines

3. Memory Allocation

4. User Interrupt Routines

5. User Interface Routines

The remaining sections of this manual describe these divisions. References are repeatedly made to object lists, object buffer lists, and element lists. These formats can be found in Appendix A. Many external global variables are used by the various display routines. These variables are described in Appendix B. The many "defines" are detailed in Appendix C.

The general design for the display software is such that all routines, defines, and external names (UNIX modifications and system routines excluded) are archived in a system library, /lib/libv.a. The user can then include this file at compile time and have access to the entire display software for his use. As a result, the convention has been adopted that all routines and global variables begin with "vg". This avoids collision of name definition if the user avoids names starting with "vg".

II.   MODIFICATIONS TO UNIX

The memory allocation scheme of UNIX had to be modified to permit the Vector General Graphics Display System to access the display list. A real-time system call,

rtime(), has been added to make the calling process real-
time by relocating the process contiguously within a
thirty-two thousand three hundred sixty-eight byte memory
block. The memory allocation of the process is not al-
lowed to cross a memory address that is a multiple of
thirty-two thousand three hundred sixty-eight. This al-
lows the vector general display to programmatically ad-
dress any of the thirty-two thousand three hundred sixty-
eight memory addresses without resetting the extended
memory bits. The rtime() call also sets the process
priority so that the operating system will not swap the
process onto the disk. This locks the process into a
fixed memory area. The real-time process can then dynami-
cally modify the display list while the vector general is
continuously accessing the same display list via its
direct memory access channel.

Because of the requirement to lock a real-time process
into a fixed memory area, the operating system permits
only a limited number of simultaneous real-time processes.
If the required memory area is allocated to another non-
swapable process or the maximum number of real-time
processes has been previously allocated, the requested
process is not made real-time and an error is returned to
the caller.

The vector general has been divided into six minor
devices. This simplifies the communication between the
user and the device driver. Each minor device has associ-
ated with it a flag that is set in response to the user's
open() command. Since the vector general is a dedicated
device, any attempt to share the vector general between
users or use the vector general without opening all minor
devices will result in an error to the caller. After all
six minor devices have been opened, a single vector gen-
eral system flag, vglock, is set. This system lock is
checked prior to any read or write operation. In addition
the process is made real-time, the user process number is
retained in vgproc, and the display is flagged as idle.
The user process number is used to pass the vector general
interrupts to the user via psignal(). The display idle
flag allows the initial display list to start the vector
general but prevents subsequent display lists from abnor-
mally terminating a previous display list.

Initialization of user controlled parameters is usual-
ly the next requirement. When the user issues a write()
command via minor device two, VGCNTRL, the contents of the
user's buffer is interpreted as the refresh reference
count. This determines the number of vector general frame
clock interrupts permitted before reinitialization of the

display list.

The vector general subroutine stack option permits accessing non-contiguous display lists. However, the vector general must have the memory address of any non-contiguous display lists encoded within a contiguous display list. This requires the user to have access to the real PDP-11 memory addresses of his display lists. The mapping of user space to real memory addresses is accomplished by the user's issuance of a write() command followed by a read() command via minor device zero, VG. The write command stores the real PDP-11 memory address of the user buffer in the variable vgaddr. The user's read command passes the value of the real PDP-11 memory address to the user via the passc() routine.

When dynamically modifying display lists, the memory address encoded in a display must often be converted to a user space address for referencing a user's display list. This capability is provided by a read() command using minor device five, VGCNVT. The base block number of the user's process obtained by the vgstrategy() routine is passed to the user via passc().

When the user has created the display, he must pass the address of the display list to the vector general. The user's write() command using minor device one, VGDISP, accomplishes this task. The lower sixteen bits of the address are stored in baddrl. The upper two bits are encoded in baddrx. If the display has not been

initialized, the routine vgstart() sends the address to the vector general and starts the vector general's operation. If the display has been initialized, the new address in baddrl represents a new display list to be used after completing the active display list.

The vector general operates independent of the user process after being given a display list. However, communication with the system routines is maintained via interrupts. A frame clock interrupt signal is generated every 8.33 milliseconds and a device interrupt is generated whenever the user operates one of the enabled peripheral devices. The frame clock interrupt increments a counter until the counter equals the refresh reference count. At that time the display is reinitiated by a call to vgstart().

The device interrupt handler passes the interrupt to the user via signal number two or fifteen. All device interrupts except the ASCII CNTRL T character are passed to the user via signal fifteen. The ASCII CNTRL T character input via the vector general keyboard is interpreted as a control key terminating the process, causing the display to be cleared, and notifying the user via signal two. Occasionally, the user may desire the CNTRL T character as data. This is provided by input of the ESC character from the vector general keyboard. This results in the next interrupt being sent to the user regardless of the type or content. Each device interrupt also causes

the values of various vector general registers to be ex-
tracted and stored in vgbuf[] for transfer to the user.

   The transfer of the vector general register values to
the user can be accomplished in several different ways.
The most common is in response to a device interrupt. The
user can acquire the interrupt state of the vector general
(except the dial positions) by issuing a read() command
using minor device one, VGDISP. The contents of vgbuf[]
are passed to the user via the passc() routine. The
user's read() command with minor device two, VGCNTRL, will
force an update of the vgbuf[] before passing the values
to the user. When minor device three, VGFNSW, is used in
a read operation, the values of the function switches are
updated prior to sending vgbuf[] to the user. VGDIAL,
minor device four, extracts the values of the ten dial
positions before transmission of vgbuf[]. The vector gen-
eral dial positions are acquired via a separate read()
command because sixteen microseconds are required to read
each dial position to the full twelve bit precision.

   The vector general's P-bit interrupt is not utilized
in this software interface. Therefore, the P-bit inter-
rupt handler is an empty routine.

   The source code for the system routines is maintained
in /usr/sys/dmr/vgdrvr.c. A copy of vgdrvr.c is included
as Appendix D.

IV. MEMORY ALLOCATION

A. FIXED MEMORY ALLOCATION

The user space memory allocation for the vector general registers and the picture display list use an unorthodox technique to obtain sequential memory locations. All the integer variables defined in the file vgreg.h correspond to the named vector general registers. The order of the variables is the order in which the values are read from the vector general. Because the loader assigns variables sequentially, this technique allows the variables to be referenced by incrementing an address pointer without requiring a structure definition.

The file vgsys.h contains a similiar sequence of integer variables used as the display list for initializing the picture parameters. Each integer variable of this file is a vector general command or a vector general data word. The order of the variables cannot be changed without affecting the operation of the vector general.

B. TUNABLE MEMORY ALLOCATION

The arrays and vectors that directly limit the size of the interface data structure are defined in the files vgglob.h and vgobj.h. When specific applications require

the interface data structure to be altered, the define
statements used to determine all array and vector sizes
are located in vgdef.h. This permits any of the system
parameters to be modified by referencing a single file.

V. USER INTERRUPT ROUTINES

A. INTERRUPT HANDLER (vgpiv)

The basic interrupt handler, vgpiv(), is called in
response to signal fifteen from the system device inter-
rupt handler. To determine which vector general device
caused the interrupt, vgpiv() obtains the interrupt state
of the vector general via the user interface routine
vgpio(). vgpio() transfers the values of eighty-three
vector general registers into eighty-three contiguous
words of memory starting at the address of vgefs1. The
value of the priority interrupt request register (PIR) can
then be examined to determine the appropriate interrupt
handler.

B.   KEYBOARD CHARACTER INTERRUPT HANDLER (vgkpiv)

The keyboard character interrupt handler, vgkpiv(), is
called when the PIK bit of the PIR register is set.   The
ASCII keyboard character, vgkbr, is placed in a circular
queue, vgkque, and the input character flag, vgkptr, is
incremented to the next cell in the queue. A character
input flag, vgkflag, is incremented each time a character
is input. The routine vggetcar() uses this flag to deter-
mine if a character has been input.

Whenever the ASCII character CNTRL P is detected,
vgkque is cleared and the flags vgkflag and vgkquefl are
reset to zero. This effectively clears the input queue of
all previous characters.

C.   MANUAL INTERRUPT HANDLER (vgmpiv)

The manual interrupt pivot, vgmpiv(), handles the PIS
interrupt from the vector general. The sole action of the
interrupt handler is to increment a counter, vgmanint.
This counter may be interrogated and cleared by the user.

D.   LIGHT PEN INTERRUPT HANDLER (vglpiv)

The light pen interrupt handler, vglpiv(), is respon-
sible for both the light pen interrupt (PIP) and the light

14

pen sense switch (SP1) interrupts. Each interrupt call to this routine must be processed and cleared before another light pen interrupt may be accepted. For each accepted interrupt the following vector general registers are stored in successive words of volpbuf[]:

Priority interrupt register (PIR)

Instruction register (IR)

Word count (WCR)

X, Y, Z coordinate registers (XR, YR, ZR respectively)

Pen resolution byte (PENR)

Whenever a light pen interrupt occurs and the light pen sense switch is processed, the counter volpsflg is incremented. The user is responsible for the use of this counter.

E. PROCESS TERMINATION ROUTINE (vgcrash)

vgcrash() is called in response to signal two. The vector general is closed via vgterm() and the process is terminated.

The user interface software has been designed to make
the detailed operation of the vector general transparent
to the user. However, the user should be familiar with
the data structure constructs used to implement the inter-
face.

The basic concept of the user interface software is to
define high level constructs which the user interface
routines convert into vector general commands. There are
three classes of constructs defined: objects, elements,
and the picture. An object is the lowest level construct
which can be displayed alone. Each object is independent-
ly rotatable, scalable, and translatable into any portion
of a thirty inch by thirty inch picture space. An object
can be as large as fifteen inches by fifteen inches and be
rotated or positioned to the extreme limits of the picture
space without distortion to any of the remaining visible
portion. Each object is composed of one or more indepen-
dently light pen hookable elements. An element is com-
posed of a series of user drawn images or characters en-
tirely relative to the untransformed image space of its
object. An object can be defined unrotated in such a way
as to fill the entire object space and then be scaled,
rotated, and moved so that the image space is the ap-
propriate size, is viewed from the appropriate aspect, and

is in the appropriate area of the picture. The picture
defines the picture scale and screen coordinates for all
objects. Figure 5-1 provides a graphic representation of
the relationship between each construct.

The user is responsible for the generation and content
of each element. Prior to its inclusion within the
display list, the user must fill each element with the
necessary draw and move commands. In addition, the user
must provide three unused words succeeding the draw-move
commands. These three words are used by the interface
routines to ensure each element is properly terminated.
This prevents the vector general from accessing memory
outside the display list if the user fails to properly
terminate the display list.

The generation and content of all objects and the pic-
ture is the responsibility of the interface software. A
set of routines are provided to link elements to objects
and objects to the picture. Dynamic modification of ob-
jects and picture parameters is also provided. However,
it is the user's responsibility to dynamically modify the
element content.

The following routines are normally transparent to the
user and should not be accessed directly by the user.

vgcntrl(addr)   vgmpiv()

vgcrash()   vgobjmod(num,fields,action)

vgconvt(abp)   vgoinit()

vgdpiv()   vgopen()

17

vgelemod(num,fields,action) vgpicmod(field,action)

vgkpiv()   vgpio(bp,mode)

vgmpiv()

The routines that are directly accessable by the user
for manipulation and modification of the display data
structure are:

vgaddele(abp,num,size)   vginit()

vgblink(type,num,action)   vgiofset(num,val)

vgclock(rate)   vgiscal(num,val)

vgcoord(num,x,y,z)   vglamps(abp)

vgcsr(num,val)   vglpen(type,num,action)

vgdelele(num)   vgmkobj()

vgdelobj(num)   vgpicture()

vgdial(abp)   vgpost(px,py)

vggetcar()   vgpscal(val)

vggetfsw(abp)   vgrotate(num,x,y,z)

vggetlpn(abp)   vgterm()

A. SUPPORTING ROUTINES

The function and operation of the routines vgcrash(),
vgdpiv(), vgkpiv(), vglpiv(), and vgmpiv() have been dis-
cussed earlier. See section IV for a discussion of their
function.
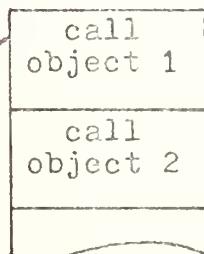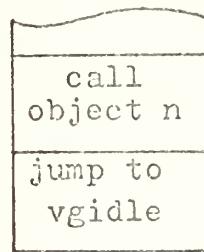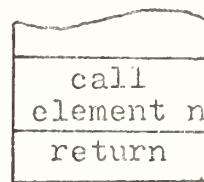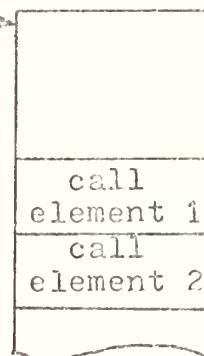
Figure 5-1. Data Structure of the Display Interface.

1. Get Real Address Routine (vgcntrl)

Since the vector general uses a DMA channel for
display list access, all noncontiguous display lists must
be linked by a real PDP-11 address at the point of discon-
tinuity. This routine takes the contents of the input
parameter, addr, and converts it to a real PDP-11 address
in a format acceptable to the vector general. The address
is passed to the system routines by the routine vgpio().
The same routine is called to return the real address in
the variable raddr. The returned address is not accept-
able to the vector general. The bit manipulation neces-
sary to convert the real address into the vector general
memory address register (MAR) format is shown in Figure
5-2.

2. Get User Space Address Routine (vgconvt)

The interface routines maintain no record of the
user's element addresses. Therefore, when a display con-
struct is modified, the user space address of the elements
involved must be obtained. The MAR format address found
in the active display list is converted back to a user
space address by this routine. The contents of the param-
eter abp (a MAR format address), is first converted back
to a real PDP-11 address (see Figure 5-2). Since the
display process is real-time and locked in memory, the

base address of the process remains constant. A call to
vgpio() returns the process block number in the variable
base. The block number is then converted to the process
base address by multiplying by sixty-four (sixty-four
bytes per block). The difference between the process base
address and the real PDP-11 address is the user space
address offset by the size of the user vector. The user
vector is a structure containing all the per process data
that does not need to be referenced while the process is
swapped. Subtraction of the user vector offset yields the
user space address.

PDP-11 WORD

VECTOR GENERAL WORD
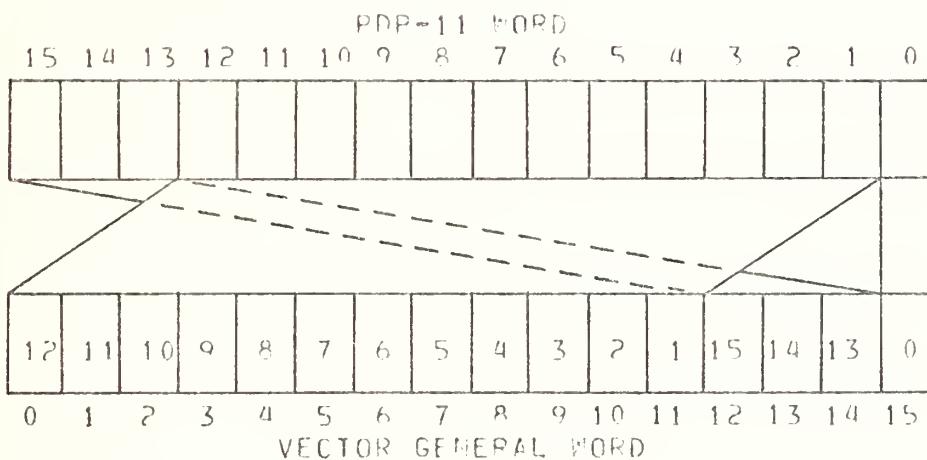
The numbers within the vector general word are the bit
numbers of the PDP-11 word.

Figure 5-2. PDP-11 to Vector General MAR Format.

3.   Modify Element Routine (vgelemod)

The only dynamic system modifications that can be per-
formed on an element is to set or clear the blink mode and
light pen hookability.

The element defined by the contents of the parameter,
num, is located by a sequential search of elements.   The
name byte field is compared with the input parameter, num.
If a match is found, the value of the field bits are OR'ed
or AND'ed into the element depending upon the value of the
parameter, action. If action is a zero, the field bits
are OR'ed into the element. If action is a one, the com-
plement of the field bits are AND'ed into the element.

The routines possible return codes are:

  0 - Normal return

 -2 - The element described in the input parameter does
     not exist

 -4 - The value of the element number is non-positive or
     greater than two hundred fifty-six, the maximum
     number of elements permitted by the name byte field
     of the vector general.

The error codes are chosen to be consistent throughout all
the interface routines.

4. Modifiy Object Routine (vaobjmod)

Object modification is more complex than the element modification, because of the number of parameters that may be varied. The structure member, vanum, of each object is compared with the input parameter, num, until a match is found. There is no requirement the object exist in the active display list. If, however, the object is part of the active display, a copy of the object is made into vgworkbuf[]. vgworkbuf[] is then linked to the active display list while the orginal object is modified.

The bit values of the input parameter, fields, determine which parameters are to be modified. The octal number representing the bit position of field and the resulting operation is as follows:

0 - Use the contents of vgf←rot[] as the new values of the rotation matrix.

01 - The value of vgf←csr is taken as the new value of the coordinate scale register.

02 - The values of vgf←dxr, vgf←dyr, and vgf←dzr are assumed to be the current values of the X, Y, and Z coordinates respectively.

010 - The new intensity offset value is obtained from vgf←ior.

020 - vgf←isr is the new value of the intensity scale.

0400 - The light pen halt interrupt is set or cleared

depending upon the value of the input parameter, action.

010000 - Display blink for the object is set or cleared as determined by the value of action.

020000 - The light pen interrupt is enabled or disabled depending upon the value of action.

After all modifications have been completed, the modified display list is again linked to the active display list.

The possible return codes are:

0 - Normal return

-1 - The value of the object number parameter is non-positive or greater than NOBJ, the maximum number of objects.

-2 - The object described in the input parameter does not exist.

5. <u>Object Initialization Routine (vginit)</u>

This routine is called as part of the program initialization routine, vginit(), to initialize the object structures and link the components of the display list. Since no objects are included at program initialization, only vgidle is linked to the display list. vgidle is a vector general HALT instruction used to keep the vector general from accessing data outside of the defined display lists. Each object is intialized with vector general instructions

permitting the following dynamic modifications:

   Change the nine register rotation matrix

   Vary the twelve bit X, Y, and Z coordinate displacements

   Modify the five bit intensity offset register

   Change the twelve bit intensity scale register

Each object is treated as a subroutine of the object buffer list, vgobjlist[]. Therefore, the last executable instruction of each object is a return subroutine jump. Since the initialized object has no elements, the first element of each object is initialized to a load MAR from stack instruction (044016). This instruction causes the vector general to get the next display instruction from vgobjlist[].

   6.   Open Device Routine (vgopen)

   This routine opens all six minor devices and saves the file descriptors for later use by vgpio() and vgterm(). The file descriptor variables for the various minor devices are:

   vgcmd - minor device zero

   vgdisp - minor device one

   vgctrl - minor device two

   vgfnsw - minor device three

   vgdial - minor device four

   vgcnvt - minor device five

If a device cannot be opened an error message is printed.

7. Picture Modification Routine (vgpicmod)

When the user desires to modify the blink or light pen modes of the picture, this routine performs the modification. If bit nine of the input parameter, field, is set, the blink mode of each element is modified according to the value of the input parameter, action. Bit eight and thirteen of field affect the light pen hookability of the picture.

8. PIO Execution Routine (vgpio)

vgpio() is called by all routines needing to communicate with the vector general. The input parameter, bp, is the address of the buffer used for the read or write operation. The second parameter, mode, is a coded description of the desired operation. The value and purpose of each mode is as follows:

1 - Read operation using minor device zero (CMD←READ).
   Used in conjunction with mode 2 to convert a user
   space address to a real address. The buffer, bp, is
   a pointer to a receptor for the real address.

2 - Write operation using minor device zero (CMD←WRITE).
   The address of bp is sent to the system routines for
   conversion from a user space address to a real address.
   dress.

3 - Read operation using minor device one (DISP←READ).

This read operation will return the eighty-three vector general register values as stored by the system routines to the eighty-three words beginning at address bp. Normally this mode is used to return the interrupt register value.

5 - Read operation using minor device two (CTRL←READ). Similar to mode three except the current values of the vector general registers (not the dial values) are extracted from the vector general prior to sending them to the caller.

6 - Write operation using minor device two (CTRL←WRITE). The contents of bp, refresh rate, is sent to the system routines to define the refresh reference count.

7 - Read operation using minor device three (FNSW←READ). Similar to mode three except the current values of the function switches are extracted from the vector general prior to the read operation.

8 - Unused

9 - Read operation using minor device four (DIAL←READ). Similar to mode three except the ten dial position values are extracted from the vector general prior to the read operation.

NOTE: The dial positions are analog devices. The conversion from analog to digital requires sixteen microseconds per dial.

27

10 - Unused

11 - Read operation using minor device five (CNVT←READ).
This operation is used with the real to user space
address conversion. The buffer pointer, bp, is the
receptor for the block number representing the be-
ginning of the real-time process.

12 - Unused

Prior to any read or write operation the value of bp
is checked for zero. The caller is prevented from reading
or writing using address zero. The read or write opera-
tion using address zero can cause the operating system to
fail.

B. USER ROUTINES

1. Add Element Routine (voaddele)

A user defined element is linked to a previously de-
fined object by this routine. The address of the user
element buffer is the parameter, abp. The input parame-
ter, size, is the number of bytes in the user's element
buffer.

NOTE: The user is required to provide six unused
bytes with each element. The six bytes (three
words) must succeed the draw-move commands.

28

The value of size is a byte count so as to follow the convention established for PDP-11 system calls. The byte count must also be even to satisfy the word addressing requirement of the vector general.

If the byte count is even and greater than six, a sequential search of all object structures is initiated. The structure member, vonum, is compared with the input parameter, num. When a match is found, a search of that object's elements is begun. The element search is conducted in increments of seven because seven words of the object structure are required to link each element to the object. The search key is the word having the name byte of the user element. The search is completed when the key word is zero. Before linking the element to the object, the six unused bytes (three words) of the users element buffer are assigned as follows:

Word one - Terminate character mode (024)

Word two - Terminate vector mode (015)

Word three - Load MAR from stack (044016)

The element to be added is always appended to the previously linked elements of the object. Therefore, the word following the new element is set to a load MAR from stack instruction (044016). The seven words linking the element to the object are next assigned as follows:

load NMR (020022)

element number

load MAR (040005)

29

MCR value (046201)

Store MAR in stack and mark (074216)

Load MAR (040016)

Address of element in MAR format

The value placed in the name byte field (NMR) of the element is returned to the user.

Several conditions could cause an error. The possible error codes and their meanings are as follows:

-1 - The value of the object number parameter is non-positive or greater than NObJ, the maximum number of objects.

-2 - The object described in the input parameter does not exist.

-3 - The number of previously assigned elements equal NELE, the maximum number of elements per object.

-4 - The value of the global variable, elenum, is greater than two hundred fifty-six, the maximum number the vector general name byte register can contain.

-5 - The user element buffer contains less than six bytes or the byte count is odd.

-6 - The user element buffer address is zero.

2. Add Object Routine (vgaddobj)

The object referenced by the parameter, num, is to be added to the active display list buffer, vgobjlist[]. The

object to be added is located by sequentially searching the object structure. When the structure member, vgnum, matches the input parameter, num, the desired object has been found. The address of the object is then placed in the variable ptr. Next the end of the active display list is found by multiplying vgobjlist[0] by three. This provides the base for inserting the object. Three words are required to link the object to vgobjlist[]. The base is the last of the three word group. The three word group is assigned as follows:

Store MAR in stack and mark (074216)

Load MAR (040016)

Address of the object in MAR format

The interface routines are designed to ensure that the objects in vgobjlist[] are always compact. Therefore, each object addition is at the end of the previously added objects. Since the last link in vgobjlist[] should always to vgidle, the voidle link must be reassigned immediately following the newly added object. The instruction sequence affecting this link is:

Load MAR (040016)

Address of vgidle in MAR format

There are four possible return codes for this routine.

0 - Normal return

-1 - The value of the object number parameter is non-

positive or greater than NOBJ, the maximum number of objects.

-2 - The object described in the input parameter does not exist.

-3 - The number of previously defined objects equal NOBJ, the maximum number of objects.

3. Display Blink Routine (voblink)

The display blink bit, MDB, of the MCR register is set or cleared by this routine. The value of the parameter, type, determines whether the picture, an object, or an element is to be affected. The value of the parameter, action, specifies the clear or set operation. The return codes are the same as those of vgobjmoo().

4. Set Refresh Rate Routine (vgclock)

The routine vgclock() is the only routine sending a control parameter to the system routines. The contents of the parameter, rate, (the refresh rate in hertz) is converted into an integer number representing the number of 8.33 millisecond interrupts permitted before refreshing the display. This integer must be between zero and nine. The converted value is sent to the system routines via the routine vgpio().

5. Change Coordinate Routine (vccoord)

The values of the X, Y, and Z coordinate displacements are updated from the input parameters by this routine. The x, y, and z input parameters are placed in vcf←dxr, vcf←dyr, and vcf←dzr respectively. The routine vgobjmod() is then called to update the coordinate displacement values. The object affected by the new values is identi-fied by the input parameter, num.

The range of the X, Y, and Z coordinate displacement values is from negative two thousand forty-eight through two thousand forty-seven. The return codes for this routine are those of vgobjmod(). The return codes for this routine are the same as those of vgobjmod().

6. Change Coordinate Scale Routine (vccsr)

The coordinate scale register of an object is updated by this routine. The lower twelve bits of the parameter, val, is assigned to vcf←csr. vgobjmod() is then called to update the coordinate scale of the object given in the parameter, num. The return codes for this routine are the same as those of vgobjmod().

7. Delete Element Routine (vgdelele)

Since elements are linked to objects by adding the element address to the element field of an object, the deletion process need only delete the address link. The element to be deleted is located in exactly the same manner as in vgmodele(). In this routine, however, only the object containing the element is modified. To prevent unwanted holes within an object, the address of the last element linked to the object is assigned to the location of the element to be deleted. This deletes the desired element but leaves a duplicate copy of the element in the display list. This duplication is eliminated by changing the last active element field to a load MAR from stack instruction (044016).

The return codes for this routine are:

0 - Normal return

-2 - The element described in the input parameter does not exist

-4 - The value of the element parameter is non-positive or greater than two hundred fifty-six, the size of the vector general name byte.

8. Delete Object Routine (vgdelobj)

Deleting an object requires the address link in the object list buffer, vgobjlist[], to be cleared and the

remaining objects links to be compacted.

The object to be deleted is located by sequentially searching the object structures for the object with the structure member, vgnum, matching the input parameter, num. The object list buffer is then searched to determine if the object is currently in the active display list. If the object is currently in the active display list, the address of the last object in vgobjlist[] is copied to the address of the object to be deleted. This last object in vgobjlist[] is then deleted by a load MAR from stack instruction (044016). The structure member, vgnum, is reset to zero making the object available for further use.

The possible return codes for this routine are:

0 - Normal return

-1 - The value of the object number parameter is non-positive or greater than NOBJ, the maximum number of objects.

-2 - The object described in the input parameter does not exist.

9. Get Dial Values Routine (vgdial)

This routine obtains the vector general dial values and returns them to the caller. The twelve bit dial values are returned to the caller in a ten word buffer provided by the caller. The contents of the parameter, abp, is the beginning address of the buffer.

10.   Get Character Routine (vggetcar)

The keyboard input flag, vgkflag, set by the keyboard interrupt handler, vgkpiv(), is checked. If it is zero, a minus one is returned to the caller. If it is non-zero, vgkflag is decremented and the value of vgkquefl is used as a pointer into the circular keyboard character queue, vgkque, to fetch the ASCII character for the caller.

11.   Get Function Switch Routine (vggetfsw)

Two vector general register words, vg←fs1 and vg←fs2, are returned to the user beginning at the buffer, abp. Each bit position of the returned words is the value of one function switch. If a bit is set, the function switch has been depressed.   The first two rows of the function device are contained in vg←fs1.   The last sixteen function switches are retained in vg←fs2.

12.   Get Light Pen Interrupt Values Routine (vggetlpn)

The values of the following vector general registers are read sequentially into the buffer abp.
  vg←pir ~ priority interrupt register
  vg←ir - instruction register
  vg←wcr - word count from start of display
  vg←xr ~ twelve bit X-coordinate displacement

36

vg←yr - twelve bit Y-coordinate displacement

vg←zr - twelve bit Z-coordinate displacement

vg←penr - one bit pen hit resolution count

13.   Display List Initialization (vginit)

The   vginit()   routine   performs all display list ini-
tialization and default parameter assignment.   The   user
process is made real-time as part of the call to vgopen().
If vgopen() can not make the process real-time or   access
all   the   minor   devices,   the   user process is terminated
without further initialization.

After   successfully accessing the vector general minor
devices, all of the data structure   buffers   are   assigned
default   values   and   linked   to form a bare bones display
system.   At this point the display could be   run   and   all
interrupts would be processed.

.   The following default picture parameters   are   set   at
display initialization:

 All function switches are cleared.

The refresh rate is set to forty hertz.

The frame clock, keyboard,   and   manual   interrupts   are
    enabled.

The display is enabled.

Maximum picture scale is set.

Post X and post Y displacement values are set to zero.

14. Set Intensity Offset Routine (vgiofset)

The input parameter, val, is placed in vgf←ior and the object modification routine, vgobjmod(), is called to update the intensity offset register of the object referenced by the input parameter, num. The return codes for this routine are those of vgobjmod().

15. Set Intensity Scale Routine (vgiscal)

The input parameter, val, is placed in vgf←isr and the object modification routine, vgobjmod(), is called to update the intensity scale register of the object referenced by the input parameter, num. The return codes for this routine are those of vgobjmod().

16. Set Function Switch Lamps Routine (vglamps)

The four successive words beginning at the buffer address abp are assigned the four vector general function switch registers vgs←fs1, vgs←fs2, vgs←fs3, and vgs←fs4.

17. Set Light Pen Enable (vglpen)

The light pen hookability of an element or object is set or cleared by this routine. If the input parameter, type, is a zero, the values of the input parameters, num

38

and action, are passed to the picture modification routine, vgpicmod(). If type is a one, the parameters are passed to the object modification routine, vgobjmod(). A type of two will modify the element, num, by calling vgelemod(). The return codes are those of the modification routine called.

18. Make Object Routine (vgmkobj)

This routine initializes an object structure for display use. Before the object can be used, it must be initialized to the system default parameters. The default parameters are:

Maximum intensity offset

Constant intensity scale

One-half coordinate scale

Zero for the X, Y, and Z coordinate displacement

Zero rotation

Same interrupts as set by the vginit routine

An unused object is found by searching the object structure until an object is found with a zero assigned as the structure member, vgnum. The instructions and default parameters are assigned to the structure and an object number is assigned from the global variable vgcurobj. The caller is given the new object number as a return value.

A possible error return code is:

-$\mathfrak{Z}$ - The number of previously defined objects equal NOBJ, the maximum number of objects.

19. Start Display Routine (vapicture)

The beginning address of the display list, vgs←ldfsl, is passed to the vector general.

20. Change Post X and Post Y Displacement Routine (vgpost)

The lower twelve bits of the input parameters, px and py, are assigned to the vector general register vgs←pdxr and vgs←pdyr respectively.

21. Change Picture Scale Routine (vgpscal)

The picture scale is changed to the twelve bit value of the input parameter, val.

22. Rotate Routine (vgrotate)

The input parameters x, y, and z are the radian measure of the rotation about the X, Y, and Z axis respectively. The necessary calculations to change the nine register rotation matrix are performed here. The final values entered into the rotation matrix represent the trigometric

40

values corresponding to the requested rotation about each axis. The rotation matrix of the object referenced by the input parameter, num, is updated to reflect the new rotation.

23. Terminate The Display (vaterm)

The vector general is cleared, the minor devices are closed, and the process is made non real-time. This is the final interface call by the user.

Picture Parameter Format

Instruction	Function
0 4 0 0 0 0	Load Lamps
0 0 0 0 0 0	Lamps 0-15
0 0 0 0 0 1	Lamps 16-32, term.
0 4 0 0 2 1	Load Pic Scale
0 7 7 7 6 1	Pic Scale, term.
0 4 0 0 4 7	Load Post Disp.
0 0 0 0 0 0	X Displacement
0 0 0 0 0 1	Y Displacement, term.
0 4 0 0 5 7	Load Lit pen enable
0 0 0 0 0 3	Enable lit pen, term.
0 4 0 0 6 4	Load Lamps
0 0 0 0 0 0	Lamps 16-23
0 0 0 0 0 1	Lamps 24-31, term.
0 4 0 0 1 7	Load Stack Pointer
0 0 0 0 0 1	Stack Address, term.
0 4 0 0 1 6	Load Mem. Addr
0 0 0 0 0 1	Object Buffer Addr, term.

Object Buffer Format

Instruction	Function

Instruction	Function
0 0 0 0 0 0	Num of objects active
0 7 4 2 1 6	Store MAR in Stack & Mark
0 4 0 0 1 6	Load MAR
0 0 0 0 0 1	Object 1 address, term.
0 7 4 2 1 6	Store MAR in Stack & Mark
0 4 0 0 1 6	Load MAR
0 0 0 0 0 1	Object 2 Address, term.

Instruction	Function
0 7 4 2 1 6	Store MAR in Stack & Mark
0 4 0 0 1 6	Load MAR
0 0 0 0 0 1	Object n Address, term.
0 4 0 0 1 6	Load MAR
0 0 0 0 0 1	vgiole Address, term.

43

Object Format

Instruction	Function
0 0 0 0 0 0	Object Number
0 4 0 0 1 4	Load Intensity Offset
0 7 7 7 6 0	Intensity Offset
0 0 0 0 0 1	Intensity Scale, term.
0 4 0 0 2 5	Load Coordinate Scale
0 3 7 7 6 0	Coordinate Scale
0 0 0 0 0 0	X-Coordinate
0 0 0 0 0 0	Y-Coordinate
0 0 0 0 0 0	Z-Coordinate
0 7 7 7 6 0	Rotate X/X
0 0 0 0 0 0	Rotate X/Y
0 0 0 0 0 0	Rotate X/Z
0 0 0 0 0 0	Rotate Y/X
0 7 7 7 6 0	Rotate Y/Y
0 0 0 0 0 0	Rotate Y/Z
0 0 0 0 0 0	Rotate Z/X
0 0 0 0 0 0	Rotate Z/Y
0 7 7 7 6 1	Rotate Z/Z, term.
0 4 0 0 2 2	Load Name Byte
0 0 0 0 0 1	Element Number
0 4 0 0 0 5	Load MCR
0 4 6 2 0 1	Enable bits, term.
0 7 4 2 1 6	Store MAR in Stack & Mark
0 4 0 0 1 6	Load MAR
0 0 0 0 0 1	Element 1 Addr, term.
0 4 0 0 2 2	Load Name Byte
0 0 0 0 0 1	Element Number, term.
0 4 0 0 0 5	Load MCR
0 4 6 2 0 1	Enable bits, term.
0 7 4 2 1 6	Store MAR in Stack & Mark
0 4 0 0 1 6	Load MAR
0 0 0 0 0 1	Element 2 addr, term.

Instruction	Function
0 4 0 0 1 6	Load MAR
0 0 0 0 0 1	Element n Addr, term.
0 4 4 0 1 6	Load MAR from Stack

Element Format

Instruction	Function
0 0 0 0 0 0	User Defined Instruction

0 0 0 0 0 0	User Defined Terminate Inst.
0 0 0 0 1 5	Terminate All Vector Modes
0 0 0 0 2 4	Terminate Character Mode
0 4 4 0 1 6	Load Mar from Stack

GLOBAL VARIABLES

All static global variables used through the interface routines are contained in the four files listed here.

vgglob.h

```
 1 int vgkque[NKQUE];              //   queue holding the last NKQUE
 2                                 //   ASCII keyboard characters
 3
 4 int vgstack[NSTACK];            //   subroutine stack buffer used
 5                                 //   with VG subroutine jumps
 6
 7 int vgworkbuf[VGOBJSIZ];        //   work buffer for sys mods
 8
 9 int vglpbuf[7];                 //   buffer to hold the VG light pen
10                                 //   interrupt registers
11
12 int vgcurobj;                   //   current object number
13
14 int vgidle;
15 char vgkflag;                   //   keyboard character flag.
16                                 //   Incremented when ASCII character
17                                 //   is input
18
19 char vgkptr;                    //   pointer to kque location
20                                 //   receiving the next
21                                 //   keyboard character
22
23 char vgkquefl;                  //   kque pointer to the next character
24                                 //   to be read
25
26 char vgmanint;                  //   manual interrupt counter
27
28 char vglpflag;                  //   light pen flag.  Set when
29                                 //   a light pen interrupt occurs.
30                                 //   Must be cleared by user before
31                                 //   a second interrupt will be
32                                 //   processed
33
34 char vglpsflg;                  //   light pen sense switch flag.
35                                 //   Set when a light pen interrupt
36                                 //   occurs and the sense switch is
37                                 //   depressed.
38
39
40 //   system work buffer used to hold the values to be
41 //   associated with the vector general register commands
42
43 int vgf_pscal;         //   picture scale
44 int vgf_rot[9];        //   rotation matrix
45 int vgf_ior;           //   intensity offset
46 int vgf_isr;           //   intensity scale
47 int vgf_csr;           //   coordinate scale
48 int vgf_dxr;           //   X-coordinate
49 int vgf_dyr;           //   Y-coordinate
50 int vgf_dzr;           //   Z-coordinate
```

```
 1  int vgelenum;                          //   current element number
 2  int vgobjlist[OLISTSIZ];               //   object list buffer
 3
 4
 5  //    structure of objects that the user has available for
 6  //    display
 7
 8  struct vgobj
 9       {
10        int vgnum;                       //   object number
11        int vglior;                      //   load intensity offset
12        iut vgior;                       //   intensity offset
13        int vgisr;                       //   intensity scale, terminate
14        int vglcsr;                      //   load coordinate scale
15        int vgcsr;                       //   coordinate scale
16        int vgx;                         //   X-coordinate
17        int vgy;                         //   Y-coordinate
18        int vgz;                         //   Z-coordinate
19        int vgrot[9];                    //   rotation matrix
20        int vgele[ELISTSIZ];             //   display element buffer
21        int vgnoop;                      //   no op instruction
22        } vgobj[NOBJ];
```

```
 1  #
 2
 3  //    NOTE:    DO NOT ALTER THE ORDER OF THESE VARIABLES
 4
 5  //    The variables in this file correspond to the named
 6  //    vector general registers.  The order is the order in which
 7  //    the values are assigned by the device driver.  The loader
 8  //    assigns these variables sequentially.  Programmatically,
 9  //    these are treated as a vector.
10
11
12        int vg_fs;                       //   function switch unit 1
13        int vg_kbr;                      //   keyboard character
14        int vg_tix;                      //   tablet x input
15        int vg_tiy;                      //   tablet y input
16        int vg_pir;                      //   priority interrupt requests
17        iut vg_mcr;                      //   mode and control (incl int enables)
18        int vg_ir;                       //   display instruction
19        int vg_wcr;                      //   word count
20        int vg_xr;                       //   X-coordinate
21        int vg_yr;                       //   Y-coordinate
22        int vg_zr;                       //   Z-coordinate
23        int vg_air;                      //   auto-increment
24        int vg_ior;                      //   intensity offset (dimming)
25        int vg_isr;                      //   intensity scale (cueing)
26        iut vg_mar;                      //   memory fetch address
27        int vg_spr;                      //   stack pointer
28        int vg_tgr;                      //   temp. general purpose
29        int vg_psr;                      //   picture scale
30        int vg_nmr;                      //   name byte
31        int vg_csr;                      //   coordinate scale
32        int vg_dxr;                      //   coordinate X displacement
33        int vg_dyr;                      //   coordinate Y displacement
34        int vg_dzr;                      //   coordinate Z displacement
35        int vg_r11r;                     //   rotation matrix X/X scale
36        int vg_r12r;                     //   rotation matrix X/Y scale
37        int vg_r13r;                     //   rotation matrix X/Z scale
38        int vg_r21r;                     //   rotation matrix Y/X scale
39        int vg_r22r;                     //   rotation matrix Y/Y scale
40        int vg_r23r;                     //   rotation matrix Y/Z scale
```

```
41        int vg_r31r;                    //    rotation matrix Z/X scale
42        int vg_r32r;                    //    rotation matrix Z/Y scale
43        int vg_r33r;                    //    rotation matrix Z/Z scale
44        int vg_wmcr;                    //    window mode control
45        int vg_xhr;                     //    window boundry X high
46        int vg_xlr;                     //    window boundry X low
47        int vg_yhr;                     //    window boundry Y high
48        int vg_ylr;                     //    window boundry Y low
49        int vg_zhr;                     //    window boundry Z high
50        int vg_zlr;                     //    window boundry Z low
51        int vg_pdxr;                    //    post X displacement
52        int vg_pdyr;                    //    post Y displacement
53        int vg_ccr;                     //    color control
54        int vg_un1;                     //    unused
55        int vg_un2;                     //    unused
56        int vg_un3;                     //    unused
57        int vg_un4;                     //    unused
58        int vg_pirx;                    //    ex dev priority interr requests
59        int vg_mcrx;                    //    ex dev interrupt enables
60        int vg_penr;                    //    pen hit resolution count
61        int vg_un5;                     //    unused
62        int vg_un6;                     //    unused
63        int vg_un7;                     //    unused
64        int vg_fs2;                     //    function switches unit 2
65        int vg_kb2;                     //    keyboard character unit 2
66        int vg_un8;                     //    unused
67        int vg_un9;                     //    unused
68        int vg_fs3;                     //    function switches unit 3
69        int vg_kb3;                     //    keyboard character unit 3
70        int vg_un10;                    //    unused
71        int vg_un11;                    //    unused
72        int vg_fs4;                     //    function switches unit 4
73        int vg_kb4;                     //    keyboard character unit 4
74        int vg_un12;                    //    unused
75        int vg_un13;                    //    unused
76        int vg_pX;                      //    picture X coordinate
77        int vg_pY;                      //    picture Y coordinate
78        int vg_pZ;                      //    picture Z coordinate
79        int vg_Jx;                      //    Joystick X input
80        int vg_Jy;                      //    Joystick Y input
81        int vg_Jz;                      //    Joystick Z input
82        int vg_dial[10];                //    dial inputs
83        int vg_cxr;                     //    window acquisition X coordinate
84        int vg_cyr;                     //    window acquisition Y coordinate
85        int vg_czr;                     //    window acquisition Z coordinate
```

vgsys.h

```
 1  #
 2  //    NOTE:    DO NOT ALTER THE CONTENTS OF THIS VECTOR!
 3
 4  //    These variables are the vector general picture initialization
 5  //    instructions and data words.  The loader assigns the variables
 6  //    sequentially allowing them to be treated as a vector.  The
 7  //    order of the variables cannot be changed without affecting
 8  //    the operation of the vector general.
 9
10
11        int vgs_ldfs1 (040000);         //    load function switch unit 1
12        int vgs_fs1 (0);                //    function switch lamp bits 0-7
13        int vgs_fs2 (01);               //    function switch lamp bits 8-15
14        int vgs_lpsr (040021);          //    load picture scale
15        int vgs_psr (077761);           //    picture scale, terminate
16        int vgs_ldpd (040047);          //    load post displacement
17        int vgs_pdxr (0);               //    post X-displacement
18        int vgs_pdyr (01);              //    post Y-displacement, terminate
19        int vgs_xmcr (040057);
20        int vgs_xp1r (03);
21        int vgs_lf2 (040064);           //    load function switch unit 2
```

```
22          int vgs_fs3 (0);              //   function switch lamps 0-7
23          int vgs_fs4 (01);             //   function switch lamps 8-15, term.
24          int vgs_lstk (040017);        //   load stack pointer
25          int vgs_stk (0);              //   stack pointer, terminate
26          int vgs_lmar (040016);        //   load memory address register
27          int vgs_mar (0);              //   memory fetch address, term.
```

The file listed here contains the defined constants used through the inter-
face routines.

vgdef.h

```
 1 #
 2
 3 /***************************************************************
 4  *   NOTE:    beware of the relation that exists between groups      *
 5  *            of defines                                             *
 6  ***************************************************************/
 7
 8 #define NELE     10                    //  max num of elements per object
 9 #define ELISTSIZ 70                    //  size of element list buffer.
10                                        //  This is equal to NELE * 7
11 #define VGOBJSIZ 89                    //  size of work buffer;
12                                        //  must be 19+ELISTSIZ
13
14 #define NOBJ     10                    //  max num of objects per picture
15 #define OLISTSIZ 33                    //  object buffer size.  This is
16                                        //  equal to (NOBJ + 1) * 3
17
18 #define NSTACK   6                     //  size of subroutine stack
19
20 #define NKQUE    6                     //  size of keyboard char queue
21
22
23
24 //  The following are the read/write defines for I/O
25
26 #define CMD_READ 1                     //  read vg commands
27 #define CMD_WRITE 2                    //  write vg commands
28 #define DISP_READ 3                    //  read display
29 #define DISP_WRITE 4                   //  write to display
30 #define CTRL_READ 5                    //  read vg controller
31 #define CTRL_WRITE 6                   //  write vg controller
32 #define FNSW_READ 7                    //  read function switches
33 #define FNSW_WRITE 8                   //  unused
34 #define DIAL_READ 9                    //  read dial positions
35 #define DIAL_WRITE 10                  //  unused
36 #define CNVT_READ 11                   //  get user base address
37 #define KYBD_WRITE 12                  //  unused
38
39
40
41 #define ROT      0                     //  rotation matirx
42 #define CSR      1                     //  coordinate scale
43 #define DXYR     2                     //  X,Y,Z coordinates
44 #define IOR      3                     //  intensity offset
45 #define ISR      4                     //  intensty scale
46 #define PSR      5                     //  picture scale
47 #define POST     6                     //  post X,Y coordinates
48 #define MPH      8                     //  light pen halt
49 #define MDB      9                     //  display blink
50 #define MEP      13                    //  enable light pen interrupt
51 #define P1P      5                     //  light pen interrupt
52 #define SP1      0                     //  light pen sense switch
53 #define PIK      3                     //  keyboard interrupt
54 #define P1S      2                     //  manual interrupt
55
56 #define CLEAR    0                     //  clear flag
57 #define SET      1                     //  set flag
58 #define P1C      0                     //  Picture type
59 #define OBJ      1                     //  object type
60 #define ELE      2                     //  element type
```

APPENDIX D
DEVICE DRIVER

Described here is the system device driver maintained as part of the
operating system. The interrupt service routines for the vector general
and the service routines for the open() and close() system calls are con-
tained in this routine.

```
 1 #
 2 #include "../param.h"
 3 #include "../conf.h"
 4 #include "../user.h"
 5 #include "../buf.h"
 6 #include "../proc.h"
 7 #include "../seg.h"
 8 #include "../systm.h"
 9
10 #define VG        0             //  minor device 0
11 #define VGDISP    1             //  minor device 1
12 #define VGCNTRL   2             //  minor device 2
13 #define VGFNSW    3             //  minor device 3
14 #define VGDIAL    4             //  minor device 4
15 #define VGCONVT   5             //  minor device 5
16 #define CLOSE     0             //  restrict access flag
17 #define OPEN      1             //  permit access flag
18 #define VGADDR    0167770       //  PDP-11 address of VG
19 #define SAR0      0             //  VG reg command
20 #define SAR1      1             //  VG reg command
21 #define SAR52     064           //  VG reg command
22 #define SAR70     0106          //  VG reg command
23 #define AKC       040000        //  ack. frame clock interrupt
24 #define SCL       01000         //  stop & clear disp controller
25 #define AKWCSD    0176600       //  ack all interpt & reset disp
26 #define BUSY      1
27 #define IDLE      0
28
29 //   four I/O channels in the PDP-11
30
31 struct {
32     int ddo;                    //  direct data output
33     int piox;                   //  extended prog I/O
34     int ma;                     //  memory address
35     int pio;                    //  prog I/O
36     };
37
38 struct buf rvgbuf;
39
40
41 char vglk [0];                  //  minor device 0 lock
42 char displk [0];                //  minor device 1 lock
43 char ctrllk [0];                //  minor device 2 lock
44 char fnswlk [0];                //  minor device 3 lock
45 char diallk [0];                //  minor device 4 lock
46 char keybdlk [0];               //  minor device 5 lock
47 char vglock [0];                //  VG system lock
48 char display;                   //  display active flag
49 char esc [0];                   //  ASCII escape flag
50 int vgbuf[80];                  //  VG register buffer
51 int clockcnt;                   //  frame clock count
52 int clockref [3];               //  frame clock ref count
53 int base;                       //  proc base address
54 int baddrl;                     //  display base address
55 int baddrx;                     //  extended memory bits
56 int i;
57 int vgaddr;                     //  buffer address
58 int vgcore;                     //  real core address
59 int *vgproc;                    //  pointer to real time process
```

```
60
61
62 vgopen(dev,flag) {
63     switch (dev.d_minor) {
64         case VG: {                      //  channel for virtual to real
65             if (vglk == OPEN) {         //  address conversion
66                 u.u_error = EIO;
67                 return;
68             }
69             vglk = OPEN;
70             break;
71         }
72         case VGDISP: {                  //  channel for display lists
73             if (displk == OPEN) {
74                 u.u_error = EIO;
75                 return;
76             }
77             displk = OPEN;
78             break;
79         }
80         case VGCNTRL: {                 //  channel for driver control
81             if (ctrllk == OPEN) {
82                 u.u_error = EIO;
83                 return;
84             }
85             ctrllk = OPEN;
86             break;
87         }
88         case VGFNSW: {                  //  channel to obtain function
89             if (fnswlk == OPEN) {       //  switches
90                 u.u_error = EIO;
91                 return;
92             }
93             fnswlk = OPEN;
94             break;
95         }
96         case VGDIAL: {                  //  channel to obtain dial
97             if (diallk == OPEN) {       //  positions
98                 u.u_error = EIO;
99                 return;
100            }
101        diallk = OPEN;
102        break;
103        }
104    case VGCONVT: {                     //  channel for real to virtual
105        if (keybdlk == OPEN) {  //  address conversion
106            u.u_error = EIO;
107            return;
108        }
109        keybdlk = OPEN;
110        break;
111    }
112    default: {
113        u.u_error = EIO;
114        return;
115    }
116    }
117if (displk && ctrllk && fnswlk && diallk && keybdik && vglk)
118    {
119    vgproc = u.u_procp;
120 // if(srtime(0) != 0)                   //  make process real time
121 //     {
122 //     vgclose();
123 //     u.u_error = EACCES;
124 //     return;
125 //     }
126    VGADDR->pio = SCL|AKC;               //  clear & reset VG
127    vglock = OPEN;                       //  enable VG system
128    display = IDLE;
129    }
130}
131
132
133
134 //   The Vector General is a dedicated device.  Therefore,
```

```
135 //   If one minor device is closed access to all minor devices
136 //   is restricted.
137
138 vgclose(dev) {
139      vglk = CLOSE;
140      vglock = CLOSE;
141      displk = CLOSE;
142      ctrllk = CLOSE;
143      fnswlk = CLOSE;
144      diallk = CLOSE;
145      keybdlk = CLOSE;
146      VGADDR->pio = SCL;              //   clear 8 reset VG
147      nonrtime();                     //   make process non real time
148      }
149
150
151
152
153
154
155 vgstrategy(abp)
156      struct buf *abp; {
157      register struct buf *bp;
158
159      bp = abp;
160      vgcore = bp->b_addr;            //   save real address of buf
161      if(display == IDLE)
162          {
163          base = vgproc->p_addr;      //   save base address for real
164                                      //   to virtual address conversion
165
166          switch(bp->b_xmem)          //   set extended memory bits for VG
167              {
168              case 00:                //   0-32k address block
169                  {
170                  baddrx = 0;
171                  break;
172                  }
173              case 01:                //   32-64k address block
174                  {
175                  baddrx = 024;
176                  break;
177                  }
178              case 10:                //   64-96k address block
179                  {
180                  baddrx = 050;
181                  break;
182                  }
183              case 11:                //   96-128k address block
184                  {
185                  baddrx = 074;
186                  break;
187                  }
188              }
189          }
190      u.u_count = 0;                  //   make sys believe I/O complete
191      iodone(bp);
192      }
193
194
195
196
197
198
199 vgwrite(dev,flag) {
200      if (vglock == CLOSE) {
201          u.u_error = EBADF;
202          return;
203          }
204      switch (dev.d_minor) {
205          case VGDISP: {              //   send display list to VG
206              physio(vgstrategy,&rvgbuf,dev,B_WRITE);
207              baddrl = vgcore;
208              if(display == IDLE) {
209                  display = BUSY;
```

```
210                         vgstart();
211                         }
212                     return;
213                     }
214             case VG: {                      // save real address
215                     physio(vgstrategy,&rvgbuf,dev,B_WRITE);
216                     vgaddr = vgcore;
217                     return;
218                     }
219             case VGCNTRL: {                 // set user refresh rate
220                     clockref = cpass();
221                     return;
222                     }
223             default: {
224                     u.u_error = EIO;
225                     return;
226                     }
227             }
228      }




vgpassc(abp)                                // pass data from here to user
239  vgpassc(abp)                               // pass data from here to user
240      int abp;                               // NOTE:  user is responsible
241      {                                      // for correct byte count
242      char *bp;
243      bp = abp;
244      while(passc(*bp) >= 0) bp++;
245      }






vgread(dev,flag) {
252  vgread(dev,flag) {
253      if (vglock == CLOSE) {
254          u.u_error = EBADF;
255          return;
256          }
257      switch (dev.d_minor) {
258          case VGCNTRL: {                     // read VG registers
259              VGADDR->pio = SAR9;
260              for (i=0;i<80;i++) vgbuf[i] = VGADDR->pio;
261              break;
262              }
263          case VGFNSW : {               // read VG function switch registers
264              VGADDR->pio = SAR0;
265              vgbuf[0] = VGADDR->pio;
266              VGADDR->pio = SAR52;
267              vgbuf[52] = VGADDR->pio;
268              break;
269              }
270          case VGDIAL: {           // read VG dial postions
271              VGADDR->pio = SAR70;
272              for (i=70;i<89;i++)
273                  {
274                  if (i<= 100);    // waste time
275                  vgbuf[i] = VGADDR->pio;
276                  }
277              break;
278              }
279          case VGDISP: {                  // read last update of VG registers
280              break;
281              }
282          case VGCONVT: {                 // send real basea address to user
283              vgpassc(&base);
284              return;
```

54

```
285                       )
286           case VG: (                    //   send real address to user
287                vgpasse(8vgaddr);
288                return;
289                )
290           default: (
291                u.u_error = E10;
292                return;
293                )
294           )
295      vgpasse(vgbuf);                     //   send VG registers to user
296      return;
297      )
298
299
300
301
302
303
304 vgstart() (
305      clockcnt = 0;
306      VGADDR->piox = baddrx;              //   set VG extended memory bits
307      VGADDR->ma = baddr1;                //   send VG display address
308      VGADDR->pio = AKWCSD;               //   start VG operation
309      )
310
311
312
313 //   VG frame clock interrupt handler.  When enabled frame
314 //   clock interrupts occur every 8.33 msec.
315
316 vgclock() (
317      if(vglock == CLOSE) return;
318      VGADDR->pio = AKC;                  //   ake frame clock intrp
319      if (++clockcnt == clockref) (       //   refresh the display
320           VGADDR->pio = SCL;
321           vgstart();
322           )
323      )
324
325
326
327
328
329 //   The current VG interface software makes no use of the
330 //   P-bit.  However, the P-bit interrupt handler is required
331 //   for system compatibility.
332
333 vgpbit() (
334      )
335
336
337
338
339
340 //   VG device interrupt handler.  Called whenever
341 //   the VG keyboard is depressed, the manual interrupt switch
342 //   is depressed, or a light pen interrupt is detected by
343 //   the VG.
344
345 vgdev()
346      (
347      if (vglock == CLOSE) return;
348      VGADDR->pio = SAR0;                 //   get VG interrupt state
349      for(i=0; i<11; i++) vgbuf[i] = VGADDR->pio;
350      VGADDR->pio = SAR52;
351      vgbuf[52] = VGADDR->pio;
352      if(!esc)
353           (
354           if(vgbuf[4]8010 88 vgbuf[1] == 012000)
355                (
356                psignal(vgproc,2);             //   terminate the process
357                goto ake;
358                )
359           if(vgbuf[4]8010 88 vgbuf[1] == 015400)
```

```
360                 {
361                 esc++;
362                 goto akc;
363                 }
364         psignal(vgproc,15);              //  send the user the interrupt
365         }
366     else esc--;
367 akc:
368     VGADDR->pio = vgbuf[4] << 3;          //  ackowledge interrupt
369     }
```

Contained here are the interface routines grouped by files. The files are in alphabetical order.

vgele.c

```
 1  #include "vgdef.h"
 2  #include "vgglob.h"
 3  #include "vgobj.h"
 4
 5
 6
 7
 8  /*                                                              *
 9   *   Add an element to an object.  Possible return codes are:   *
10   *           -1   illegal object number                         *
11   *           -2   nonexistant object                            *
12   *           -3   object cannot access more elements            *
13   *           -4   element number is out of range                *
14   *           -5   user display buffer is less than 6 bytes      *
15   *           -6   user buffer address is zero                   *
16   *       normal return is the element number                    *
17   *                                                              */
18
19  vgaddele(abp,num,size)
20      int abp;                            //   user element buffer
21      int num;                            //   object number
22      int size;                           //   num bytes in user buffer
23      {
24      int i,j;
25      int *bp;                            //   buffer pointer
26
27      if((bp=abp) == 0) return(-6);           // check for buf addr of 0
28      if(size&01 || (size=>>1)<=3) return(-5);     // check byte count
29      if(num<=0) return(-1);          // check object number
30  /*                                                              *
31   *   Search all the object structures for the object.           *
32   *                                                              */
33      for(i=0;i<NOBJ;i++) if(vgobj[i].vgnum == num) break;
34      if(i >= NOBJ) return(-2);                   // object doesn't exist
35  /*                                                              *
36   *   Find the first empty element location in the object        *
37   *   structure.                                                 *
38   *                                                              */
39      for (j=1;j<ELISTSIZ;j =+ 7)
40              if (vgobj[i].vgele[j] == 0) break;
41      if(j==ELISTSIZ) return(-3);                 // no empty elements
42
43      if((++vgelenum)>256) return(-4);     // no empty elements
44  /*                                                              *
45   *   Fill the last 3 words of the element buffer.               *
46   *                                                              */
47      *(bp + (size-1)) = 044016;              // ld MAR from stack inst
48      *(bp +(size-2)) = 024;                  // term char mode
49      *(bp +(size-3)) = 015;                  // term all modes exept char
50  /*                                                              *
51   *   Fill the element words associated with the object.         *
52   *                                                              */
53      vgobj[i].vgele[j+6] = 044016;
54      vgobj[i].vgele[j+5] = vgcntrl(bp)|01;
55      vgobj[i].vgele[j+4] = 040016;
56      vgobj[i].vgele[j+3] = 074216;
57      vgobj[i].vgele[j+2] = 046201;
58      vgobj[i].vgele[j+1] = 040005;
```

```
59      vgobj[i].vgele[j]    = (vgelenum<<8)|01;
60      vgobj[i].vgele[j-1] = 040022;
61      return(vgelenum);
62      }
63
64
65
66
67
68
69
70
71
72
73  /*                                                                    *
74   *   Delete the element by searching all object structures           *
75   *   for the given element.  Possible return codes are:              *
76   *               0    normal return                                  *
77   *              -2    element does not exist                         *
78   *              -4    element number is out of range                *
79   *                                                                   */
80
81  vgdelele(num)
82      int num;                            //   element number
83      {
84      int i,j;
85
86      if(num<=0 || num>256) return(-4);    //  check element num
87  /*                                                                    *
88   *   Sequentially search all object structures and elements for the  *
89   *   object containing the element.                                  *
90   *                                                                   */
91      for(i=0;i<NOBJ;i++)
92          {
93          for(j=1;j<ELISTSIZ;j =+ 7)
94              {
95              if(vgobj[i].vgele[j] == 0) break;
96              if (vgobj[i].vgele[j] == ((num<<8)|01)) goto found;
97              }
98          }
99      return(-2);                               //   element doesn't exist
100 /*                                                                    *
101  *   Delete the element and compact the remaining elements.          *
102  *                                                                   */
103 found:
104     while(vgobj[i].vgele [j+7] != 0)
105         {
106         vgobj[i].vgele[j+6] = vgobj[i].vgele[j+13];
107         vgobj[i].vgele[j+5] = vgobj[i].vgele[j+12];
108         vgobj[i].vgele[j+2] = vgobj[i].vgele[j+9];
109         vgobj[i].vgele[j]   = vgobj[i].vgele[j+7];
110         j =+ 7;
111         }
112     vgobj[i].vgele[j-1] = 044016;
113     vgobj[i].vgele[j] = 0;
114     return(0);
115     }
116
117
118
119
120
121
122
123
124
125
126 /*                                                                    *
127  *   Modify the light pen and display blink fields of the MCR        *
128  *   register of element.  Possible return codes are:                *
129  *               0    normal return                                  *
130  *              -2    element doesn't exist                          *
131  *              -4    element out of range                           *
132  *                                                                   */
133
```

```
134
135 vgelemod(num,field,action)
136     int num;                    //   element number
137
138     int field;                  //   0400       light pen halt
139                                 //   01000      display blink
140                                 //   020000     light pen hit detect interrupt
141
142     char action;                //   0 - clear
143                                 //   1 - set
144     {
145     int i,j;
146
147     if(num<=0 || num>NELE) return(-4);   //   check element number
148 /*                                                                                *
149  *  Find the object containing the element.                                       *
150  *                                                                                */
151
152     for (i=0;i<NOBJ;i++)
153         {
154         for (j=1;j<ELISTSIZ;j =+ 7)
155             {
156             if (vgobj[i].vgele[j] == 0) break;
157             if (vgobj[i].vgele[j] == ((num<<8)|01)) goto found1;
158             }
159         }
160     return(-2);                              // element doesn't exist
161
162 /*                                                                                *
163  *  Modify the light pen and blink control.                                       *
164  *                                                                                */
165 found1:
166     if(action == SET)
167         vgobj[i].vgele[j+2] =| field;
168     else if(action == CLEAR)
169         vgobj[i].vgele[j+2] =& ~(field|0100000);
170     return(0);
171     }
```

vginit.c

```
 1 #include "vgdef.h"
 2 #include "vgglob.h"
 3 #include "vgsys.h"
 4 #include "vgobj.h"
 5
 6
 7 extern vgcrash();
 8 extern vgdpiv();
 9
10
11 /*                                                                                *
12  *  The vector general initialization routine defines all                         *
13  *  system instructions and links all system buffers.                             *
14  *                                                                                *
15  *  If the process can't become real-time or if all minor                         *
16  *  devices are unable to be opened the process is terminated.                     *
17  *                                                                                */
18
19
20 vginit()
21     {
22
23 /*                                                                                *
24  *  Open all vector general minor devices.                                        *
25  *                                                                                */
26
27     if (vgopen() != 0) vgcrash();
28
```

```
29  /*                                                                    *
30  *    Make the process real-time.  This call is placed here            *
31  *    only until the system rtime() call from the driver level         *
32  *    can be debugged.  It should be a call at the driver level         *
33  *    when all minor devices are opened.                                *
34  *                                                                      */
35
36        if(rtime(0) != 0)
37            {
38            perror("rtime error");
39            vgcrash();
40            }
41
42        signal(2,vgcrash);
43        signal(15,vgdpiv);
44
45        vgclock(40);                          //    set default refresh rate
46
47        vgs_mar  = vgoinit()|01;              //    get address of objlist
48        vgs_stk = vgcntrl(vgstack)|01;        //    stack addr in MAR format
49        vgstack[0] = vgcntrl(&vgidle)|01;     //    stack underflow protection
50        vgidle = 036000;                      //    halt instruction
51        vglpflag = 0;                         //    light pen interrupt flag
52        vglpsflg = 0;                         //    light pen sense switch flag
53        vgkflag = 0;                          //    keyboard flag
54        vgkptr = 0;                           //    keyboard queue pointer
55        vgkquefl = 0;                         //    keyboard input flag
56        return;
57        }
58
59
60
61
62
63  /*                                                                    *
64  *    fill buffer with the post X-coordinate and the post             *
65  *    Y-coordinate values.  The values that px and py may assume       *
66  *    are 0177760 (-2048) through 077760 (2047).                       *
67  *                                                                      */
68
69  vgpost(px,py)
70        int px;                               //    post X-coordinate
71        int py;                               //    post Y-coordinate
72        {
73        vgs_pdxr = px << 4;
74        vgs_pdyr = py << 4;
75        }
7
77
78
79
80
81  /*                                                                    *
82  *    Set/Clear the function switch lamps.  Each bit set in the        *
83  *    input buffer will affect one lamp.                               *
84  *                                                                      */
85
86  vglamps(abp)
87        int abp;                    //    two word buffer pointer
88        {
89        int *bp;
90        if(abp==0) return(-6);
91        bp = abp;
92        vgs_fs1 = *bp & 0177400;
93        vgs_fs2 = ((*bp & 0377) << 8)|01;
94        vgs_fs3 = *(++bp) & 0177400;
95        vgs_fs4 = ((*bp & 0377) << 8)|01;
96        }
97
98
99
100
101
102  /*                                                                    *
103  *    Modify the picture parameters.                                  *
```

```
104    *    The values field can assume are:                                    *
105    *                                                                         */
106
107
108  vgpicmod(field,action)
109       int field;                  //   0400      light pen halt
110                                    //   01000     display blink
111                                    //   020000    light pen hit detect interrupt
112
113       int action;                 //   0     clear
114                                    //   1     set
115
116       {
117       int i;
118
119       i = 0;
120       while(i<NOBJ) vgobjmod(vgobj[i++].vgnum,field,action);
121       }
122
123
124
125
126
127  /*                                                                          *
128   *    Start the display.                                                    *
129   *                                                                          */
130
131  vgpicture()
132       {
133       vgpio(&vgs_ldfs1,DISP_WRITE);                    //   start display
134       }
135
136
137
138
139
140  /*                                                                          *
141   *    Modify the picture scale.  The range of values are 0 - 1.             *
142   *                                                                          */
143
144  vgpscal(val)
145       double val;                                 //   picture scale value
146       {
147       int temp;                                   //   temp integer value
148       temp = val * 2047;
149       vgs_psr = (temp<<4)|01;
150       return(0);
151       }
```

vgintr.c

```
1  #include "vgdef.h"
2  #include "vgglob.h"
3  #include "vgreg.h"
4
5
6  /*                                                                          *
7   *    when a light pen interrupt, a sense switch interrupt, a               *
8   *    keyboard interrupt, or a manual interrupt occurs   the               *
9   *    VG system interrupt driver passes the interrupt to the               *
10  *    user via this routine.                                               *
11  *                                                                         *
12  *    NOTE:     interrupts are passed via signal 15                        *
13  *                                                                         */
14
15  vgdpiv()
16       {
17       int i;
18
```

```
19          signal(15,vgdpiv);
20          vgpio(&vg_fs,DISP_READ);                // get interrupt state from VG
21
22          for (i=0;i<7;i++) if((vg_pir>>i)&01) switch(i)
23              {
24
25              case PIP:                           // light pen interrupt
26
27              case SP1:                           // light pen sense switch
28                  {                               // interrupt
29                  vglpiv();
30                  break;
31                  }
32
33              case PIK:                           // keyboard interrupt
34                  {
35                  vgkpiv();
36                  break;
37                  }
38
39              case PIS:                           // manual interrupt
40                  {
41                  vgmpiv();
42                  break;
43                  }
44              }
45          return;
46          }
47
48
49
50
51
52 /*                                                                          *
53  *   light pen interrupt handler                                           *
54  *   store the resulting light pen interrupt values in vglpbuf             *
55  *   only if the previous interrupt has been processed by the             *
56  *   user.                                                                 *
57  *                                                                         */
58
59 vglpiv()
60      {
61      if((vglpflag==0) && (vg_pir&01))
62          {
63          vglpbuf[0] = vg_pir>>3;                 // mode & control reg
64          vglpbuf[1] = vg_ir;                     // instruction word reg
65          vglpbuf[2] = vg_wcr;                    // word count reg
66          vglpbuf[3] = vg_xr>>4;                  // X-coordinate
67          vglpbuf[4] = vg_yr>>4;                  // Y-coordinate
68          vglpbuf[5] = vg_zr>>4;                  // Z-coordinate
69          vglpbuf[6] = vg_pcur;                   // pen resolution byte
70          vglpflag++;
71          }
72      if (vg_pir&01) vglpsflg++;
73      }
74
75
76
77
78
79 /*              .                                                           *
80  *   Vector General Keyboard Character Pivot                               *
81  *   when a keyboard character interrupt has occured, this routine         *
82  *   gets the keyboard character from the Vector General                   *
83  *   places it in vgkque, and increments the character flag,               *
84  *   vgkflag, to indicate a character has been input                       *
85  *                                                                         *
86  *   NOTE:    The user should empty the queue by calling                   *
87  *   vggetcar().                                                           *
88  *                                                                         */
89
90 vgkpiv() {
91      vgkflag++;
92      if (vg_kbr == 010000)
93          {
```

```
94          vgkflag = 0;
95          vgkptr = 0;
96          vgkquefl = 0;
97          return;
98          }
99      if (vgkptr == NKQUE) vgkptr = 0;
100     vgkque[vgkptr++] = vg_kbr;
101     }
102
103
104
105
106 /*                                                                     *
107  *   Vector General Manual Interrupt Pivot.                            *
108  *   when a manual switch interrupt occurs, this routine increments    *
109  *   the manual interrupt counter                                      *
110  *                                                                     */
111
112 vgmpiv() {
113     vgmanint++;
114     }
115
116
117
118
119
120
121
122 /*                                                                     *
123  *   kill the process.                                                 *
124  *   called when a condition requires process termination             *
125  *   i.e. "rubout" on the DATAMEDIA terminal or CTRL T                 *
126  *   on the vector general keyboard                                    *
127  *                                                                     */
128
129 vgcrash() {
130     vgterm();
131     exit();
132     }
```

vgobj.c

```
1 #include "vgdef.h"
2 #include "vgglob.h"
3 #include "vgobj.h"
4
5
6
7 .
8 /*                                                                     *
9  *   Add the object to the active display list.                        *
10  *   The possible return values are:                                   *
11  *            0    normal termination                                  *
12  *           -1    object number is neg., zero, or greater than NOBJ   *
13  *           -2    object doesn't exist                                *
14  *           -3    object list is full                                 *
15  *                                                                     */
16
17 vgaddobj(num)
18     int num;                            //   object number
19     {
20     int *addr;                          //   object address
21     int l,J;
22
23     if(num<=0) return(-1);   //   check object number
24     if(vgobjlist[0]>NOBJ) return(-3);    //   object list full
25 /*                                                                     *
26  *   Find the object structure with the desired object number          *
27  *                                                                     */
```

63

```
28          for(i=0;i<NOBJ;i++)  if(vgobj[i].vgnum == num) break;
29          if(i>=NOBJ) return(-2);                  //  object doesn't exist
30
31          j = vgobjlist[0] *3;
32          vgobjlist[j+2] = vgcntrl(&vgidle)|01;
33          vgobjlist[j+1] = 040016;                 //   load MAR
34          vgobjlist[j]   = vgcntrl(&vgobj[i].vgnum)|01;
35          vgobjlist[j-1] = 040016;                 //   load MAR
36          vgobjlist[j-2] = 074216;                 //  store MAR in stack & mark
37          vgobjlist[0]++;
38          return(0);
39          }
40
41
42
43
44
45
46
47
48
49
50  /*                                                                         *
51   *    Find the object, delete it, and if in the active display list        *
52   *    compact the resulting active display list.  Possible return          *
53   *    values are:                                                          *
54   *               0    normal return                                        *
55   *              -1    object number out of range                           *
56   *              -2    object doesn't exist                                 *
57   *                                                                         */
58
59  vgdelobj(num)
60      int num;                                 //   object number
61      {
62      int addr;                                //   address pointer
63      int i,j,k;
64
65      if(num<=0) return(-1);   //   check object number
66  /*                                                                         *
67   *    Sequentially search the object structures until the desired          *
68   *    object can be found.                                                 *
69   *                                                                         */
70      for(i=0;i<NOBJ;i++)  if(vgobj[i].vgnum==num) break;
71      if(i>=NOBJ) return(-2);                  //   object doesn't exist
72      vgobj[i].vgnum = 0;                      //   delete the object
73  /*                                                                         *
74   *    Check if the object is in the active display list.  If yes           *
75   *    the link in vgobjlist must be removed.  If no then only the          *
76   *    object number must be zeroed.                                        *
77   *                                                                         */
78      addr = vgcntrl(&vgobj[i].vgnum)|01;
79      for(j=1;j<vgobjlist[0];j++)  if(vgobjlist[j*3]==addr) break;
80  /*                                                                         *
81   *    Compact the resulting display list.                                  *
82   *                                                                         */
83      if(j<vgobjlist[0])
84          {
85          vgobjlist[0]--;
86          vgobjlist[j*3] = vgobjlist[vgobjlist[0]*3];
87          vgobjlist[vgobjlist[0]*3] = vgcntrl(&vgidle)|01;
88          }
89      return(0);
90      }
91
92
93
94
95
96
97
98
99
100
101 /*                                                                         *
102  *    Find an unused object structure and initialize the structure         *
```

```
103   *    to all default parameters.  Possible return codes are:           *
104   *              -3  all objects previously defined                      *
105   *    Normal return is the object number assigned to the new object     *
106   *                                                                      */
107
108  vgmkobj()
109       {
110       int i;
111  /*                                                                      *
112   *    Find the first unused object structure.                           *
113   *                                                                      */
114       for(i=0;i<NOBJ;i++)  if(vgobj[i].vgnum==0)  break;
115       if(i>=NOBJ)  return(-3);            //   all object in use
116       vgobj[i].vgnum = ++vgcurobj;
117       vgobj[i].vgior = 077760;            //   intensity offset
118       vgobj[i].vgisr = 01;                //   intensity scale, terminate
119       vgobj[i].vgcsr = 037760;            //   coordinate scale
120       vgobj[i].vgx   = 0000000;           //   X-coordinate
121       vgobj[i].vgy   = 0000000;           //   Y-coordinate
122       vgobj[i].vgz   = 0;                 //   Z-coordinate
123       vgobj[i].vgrot[0] = 077760;         //   fill rotation matrix
124       vgobj[i].vgrot[4] = 077760;
125       vgobj[i].vgrot[8] = 077761;         //   terminate
126       return(vgobj[i].vgnum);
127       }
128
129
130
131
132
133
134
135
136
137
138  /*                                                                      *
139   *    Modify the object as indicated by the input parameter.            *
140   *    Possible return codes are:                                        *
141   *              0   normal return                                       *
142   *              -1  object number out of range                          *
143   *              -2  object doesn't exist                                *
144   *                                                                      */
145
146  vgobjmod(num,fields,action)
147       int num;                        //   object number
148
149       int fields;                     //   0 - rotation matrix
150                                       //   1 - coordinate scale
151                                       //   2 - X,Y,Z coordinates
152                                       //   3 - intensity offset
153                                       //   4 - intensity scale
154                                       //   8 - light pen halt
155                                       //   9 - display blink
156                                       //   13 - light pen interrupt
157
158       char action;                    //   0 - clear
159                                       //   1 - set
160       {
161       int ptr;                        //   pointer to object to be modified
162       int obj;                        //   ptr. to obj. in obj. buf. list
163       int i,j;
164       int taddr;                      //   temp MAR addr of object
165       int *addr;                      //   address of an object
166       char active;                    //   object in active display list
167
168       if(num <= 0)  return(-1);           //   check object number
169  /*                                                                      *
170   *    Sequentially search the object structures for the desired         *
171   *    object.                                                           *
172   *                                                                      */
173       for(ptr=0;ptr<NOBJ;ptr++)  if(vgobj[ptr].vgnum == num)  break;
174       if(ptr>=NOBJ)  return(-2);                      //   object does not exist
175  /*                                                                      *
176   *    Check the active display list for the object to be modified.      *
177   *                                                                      */
```

```
178        addr = &vgobj[ptr].vgnum;
179        taddr = vgcntrl(addr)|01;
180        for(obj=1;obj<vgobjlist[0];obj++)if(vgobjlist[obj*3]==taddr)break;
181 /*                                                                        *
182  *   Make a copy of the object and link the copy to the active           *
183  *   display list.                                                        *
184  *                                                                        */
185        active = 0;
186        if (obj < vgobjlist[0])
187             {
188             for(i=0;i<VGOBJSIZ;i++) vgworkbuf[i] = *(addr++);
189             taddr = vgobjlist[obj*3];       //  save MAR addr of the org. obj
190             vgobjlist[obj*3] = vgcntrl(vgworkbuf)|01;
191             active++;                       //  set object active flag
192             }
193 /*                                                                        *
194  *   Make the modifications to the orginal object.                        *
195  *                                                                        */
196        for(i=0;i<16;i++)
197             {
198             if ((fields>>i)&01) switch(i)
199                  {
200                  case ROT:                              //  rotation matrix
201                       {
202                       for(j=0;j<9;j++) vgobj[ptr].vgrot[j]=vgf_rot[j]<<4;
203                       vgobj[ptr].vgrot[8] =| 01;
204                       break;
205                       }
206                  case CSR:                              //  coordinate scale
207                       {
208                       vgobj[ptr].vgcsr = vgf_csr<<4;
209                       break;
210                       }
211                  case DXYR:                             //  X,Y,Z coordinates
212                       {
213                       vgobj[ptr].vgx = vgf_dxr<<4;
214                       vgobj[ptr].vgy = vgf_dyr<<4;
215                       vgobj[ptr].vgz = vgf_dzr<<4;
216                       break;
217                       }
218                  case IOR:                              //  intensity offset
219                       {
220                       vgobj[ptr].vgior = vgf_ior<<4;
221                       break;
222                       }
223                  case ISR:                              //  intensity scale
224                       {
225                       vgobj[ptr].vgisr = (vgf_isr<<4)|01;
226                       break;
227                       }
228                  case MPH:                              //  light pen halt
229                  case MDB:                              //  display blink
230                  case MEP:                              //  enable light pen
231                       {
232                       if(action==SET)
233                            for (j=1;j<ELISTSIZ;j=+ 7)
234                                 {
235                                 if(vgobj[ptr].vgele[j]==0) break;
236                                 vgobj[ptr].vgele[j+2] =| 1<<i;
237                                 }
238                       if(action == CLEAR)
239                            for(j=1;j<ELISTSIZ;j=+ 7)
240                                 {
241                                 if(vgobj[ptr].vgele[j]==0) break;
242                                 vgobj[ptr].vgele[j+2]=& ~((1<<i)|0100000);
243                                 }
244                       break;
245                       }
246                  default:
247                       {
248                       break;
249                       }
250                  }
251             }
252        if (active) vgobjlist[obj*3] = taddr;
```

```
253        return(0);
254        }
255
256
257
258
259
260
261
262
263
264
265 /*                                                                      *
266  *     initiate the object buffer list and fill all constant           *
267  *     fields.                                                          *
268  *                                                                      */
269
270  vgoinit()
271        {
272        int i;
273        int J;
274
275        J = 1;
276        vgcurobj = 0;
277        vgelcnum = 0;
278
279 /*                                                                      *
280  *     Initialize the object list buffer                               *
281  *                                                                      */
282
283        vgobjlist[0] = 1;                       // set first object number  •
284        vgobjlist[1] = 030000;                  //   halt
285
286 /*                                                                      *
287  *     Initialize each object structure                                *
288  *                                                                      */
289                                                                       •
290        for(i=0;i<NOBJ;i++)
291            {
292            vgobj[i].vglior  = 040014;          // load intensity offset reg
293            vgobj[i].vglcsr  = 040023;          // load coordinate scale reg
294            vgobj[i].vgele[0]= 044016;          // load MAR from stack
295            }
296        return(vgcntrl(vgobjlist));
297        }




                                 vgpio.c



 1 #include "vgdef.h"
 2
 3      int vgctrl;                 //    controller file descriptor
 4      int vgdials;                //    dials file descriptor
 5      int vgdisp;                 //    display file descriptor
 6      int vgfnsw;                 //    function switch file descriptor
 7      int vgcnvt;                 //    keyboard file descriptor
 8      int vgcmd;                  //    command file discriptor
 9
10 /*                                                                      *
11  *    Convert the display list address into a user space address       *
12  *                                                                      */
13
14 vgconvt(abp)
15      int abp;                    //    display list address pointer
16      {
17      int bp;
18      int base;                   //    base block number of process
19      int page;                   //    most sig 3 bits of address
20      int virtual;                //    user space address
21
```

67

```
22  /*                                                                    *
23   *    Convert the display list address into a real address            *
24   *                                                                    */
25
26       page = (abp & 016) << 12;
27       abp = (abp>>3) & 017776;
28       bp = abp | page;
29
30  /*                                                                    *
31   *    Get the base block number of the process                        *
32   *                                                                    */
33
34       vgpio(&base,CNVT_READ);
35
36  /*                                                                    *
37   *    Convert the block number into an address and convert the real   *
38   *    address into a user space address.                              *
39   *                                                                    */
40
41       virtual = (bp - (base<<06)) - 02000;
42       return(virtual);
43       }
44
45
46
47
48
49  /*                                                                    *
50   *    Convert the user space address into a real address in MAR       *
51   *    format.                                                         *
52   *                                                                    */
53
54  vgcntrl(addr)
55       int addr;                     //   user space address pointer
56       {
57       int *laddr;                   //   logical address
58       int raddr;                    //   real address
59       int mar;                      //   VG memory address register
60       laddr = addr;
61
62  /*                                                                    *
63   *    Get the real address of the user space address                  *
64   *                                                                    */
65
66       vgpio(laddr,CMD_WRITE);
67       vgpio(&raddr,CMD_READ);
68
69  /*                                                                    *
70   *    Convert the real address into MAR format.                       *
71   *                                                                    */
72
73       mar = (raddr) >> 12;
74       mar =& 016;
75       mar =| (raddr<<3);
76       return(mar);
77       }
78
79
80
81
82
83  /*                                                                    *
84   *    Open each of the minor devices and retain the file              *
85   *    descriptors. The minor device number and the file descriptor    *
86   *    associated with it is:                                          *
87   *               0    vgcmd         /dev/vg                           *
88   *               1    vgdisp        /dev/vgdp                         *
89   *               2    vgcntrl       /dev/vgct                         *
90   *               3    vgfnsw        /dev/vgfs                         *
91   *               4    vgdials       /dev/vgdl                         *
92   *               5    vgcnvt        /dev/vgkb                         *
93   *                                                                    */
94
95  vgopen()
96       {
```

68

```
97      if ((vgcmd = open("/dev/vg",2)) < 0)            // minor device 0
98          {
99          perror("open vgcmd error");
100         return(-7);
101         }
102     if ((vgdisp = open("/dev/vgdp",2)) < 0)         // minor device 1
103         {
104         perror("open vgdisp error");
105         return(-2);
106         }
107     if ((vgctrl = open("/dev/vgct",2)) < 0)         // minor device 2
108         {
109         perror("open vgctrl error");
110         return(-3);
111         }
112     if ((vgfnsw = open("/dev/vgfs",0)) < 0)         // minor device 3
113         {
114         perror("open vgfnsw error");
115         return(-4);
116         }
117     if ((vgdials = open("/dev/vgd1",0)) < 0)        // minor device 4
118         {
119         perror("open vgdials error");
120         return(-5);
121         }
122     if ((vgcnvt = open("/dev/vgkb",0)) < 0)         // minor device 5
123         {
124         perror("open vgcnvt error");
125         return(-6);
126         }
127     return(0);
128     }
129
130
131
132
133
134/*                                                                          *
135 *  Vector General Read/Write Routine                                       *
136 *  all communication with the vector general is handled via this           *
137 *  routine.   The mode determines the action to be taken.                  *
138 *  The acceptable values for the mode are:                                 *
139 *          1 - read using minor device 0                                   *
140 *          2 - write using minor device 0                                  *
141 *          3 - read using minor device 1                                   *
142 *          4 - write using minor device 1                                  *
143 *          5 - read using minor device 2                                   *
144 *          6 - write using minor device 2                                  *
145 *          7 - read using minor device 3                                   *
146 *          8 - write using minor device 3                                  *
147 *          9 - read using minor device 4                                   *
148 *          10 - write using minor device 4                                 *
149 *          11 - read using minor device 5                                  *
150 *          12 - write using minor device 5                                 *
151 *  Return of -1 is the result of an addressing error.                      *
152 *                                                                          */
153
154vgpio(bp,mode)
155     int bp;                             //  buffer address pointer
156     int mode;                           //  type I/O operation
157     {
158     int *abp;
159
160/*                                                                          *
161 *  Check buffer address.  Do not perform the operation is the              *
162 *  address is zero.                                                        *
163 *                                                                          */
164
165     if(bp == 0) vgcrash();
166
167     abp = bp;
168     switch (mode)
169         {
170         case CMD_READ:                  //  calc abs address
171             {
```

69

```
172            if(read(vgcmd,abp,2) < 0)
173                perror("CMD_READ error");
174            break;
175            }
176
177        case CMD_WRITE:                    //  get abs address
178            {
179            if(write(vgcmd,abp,2) < 0)                                              •
180                perror("CMD_WRITE error");
181            break;
182            }
183
184        case DISP_READ:                    //  get interrupt registers
185            {
186            if(read (vgdisp,abp,166) < 0)
187                perror("DISP_READ error");
188            break;
189            }
190
191        case DISP_WRITE:                   //  sent display list
192            {
193            if(write(vgdisp,abp,10) < 0)
194                perror("DISP_WRITE error");
195            break;
196            }
197
198        case CTRL_READ:                    //  get current vg register value
199            {
200            if(read (vgctrl,abp,2) < 0)
201                perror("CTRL_READ error");
202            break;
203            }
204
205        case CTRL_WRITE:                   //  sent system control words
206            {
207            if(write(vgctrl,abp,2) < 0)
208                perror("CTRL_WRITE error");
209            break;
210            }
211
212        case FNSW_READ:                    //  get func switch value
213            {
214            if(read (vgfnsw,abp,166) < 0)
215                perror("FNSW_READ error");
216            break;
217            }
218
219        case FNSW_WRITE:                   //  not used
220            {
221            break;
222            }
223
224        case DIAL_READ:                    //  get dial positions
225            {
226            if(read(vgdials,abp,166) < 0)
227                perror("DIAL_READ error");
228            break;
229            }
230
231        case DIAL_WRITE:                   //  unused
232            {
233            break;
234            }
235
236        case CNVT_READ:                    //  get base address
237            {
238            if(read(vgcnvt,abp,2) < 0)
239                perror("CNVT_READ error");
240            break;
241            }
242
243        case KYBD_WRITE:                   //  unused
244            {
245            break;
246            }
```

```
247        default:
248            {
249            break;
250            }
251        }
252    }
253
254
255
256
257
258/*                                                                    *
259 *    Terminate vector general operations.   Close all minor devices  *
260 *    and make the process non real-time.                             *
261 *                                                                    */
262
263vgterm()
264    {
265    if(close(vgdisp) < 0)
266        perror("Close error");
267    }
```

vgrdwri.c

```
 1 #include "vgdef.h"
 2 #include "vgreg.h"
 3 #include "vgglob.h"
 4
 5
 6 /*                                                                    *
 7  *    Get the ten vector general dial values.                          *
 8  *                                                                    */
 9
10 vgdial(abp)
11     int abp;                              //   ten word buffer pointer
12     {
13     int *bp;
14     int i;
15     if(abp==0) return(-6);
16     bp = abp;
17     vgpio(&vg_fs,DIAL_READ);           //   get dial values from VG
18     for(i=0; i<10; i++) *(bp++) = vg_dial[i]>>4;
19     }
20
21
22
23
24
25 /*                                                                    *
26  *    get function switch values                                      *
27  *    the thirty-two values of the VG function switches               *
28  *    are returned to the user via the two word buffer.               *
29  *    A bit that is set corresponds to a VG function switch            *
30  *    that has been depressed.                                        *
31  *                                                                    */
32
33 vggetfsw(abp)
34     int abp;                              //   2 word buffer
35     {
36     int *bp;
37     if(abp==0) return(-6);
38     bp = abp;
39     vgpio(&vg_fs,FNSW_READ);           // get function switch values
40     *(bp++) = vg_fs;
41     *bp = vg_fs2;
42     }
43
44
45
```

71

```
46
47
48  /*                                                                    *
49   *   Calculate the refresh rate and sent the value to the vector      *
50   *   general.  The value sent must be an integer between 0 and 9      *
51   *                                                                    */
52
53  vgelock(rate)
54      int rate;                                    //  refresh rate in hertz
55      {
56      rate = 120/rate;
57      vgpio(&rate,CTRL_WRITE);
58      }
```

vgusr.e

```
 1  #include  "vgdef.h"
 2  #include  "vgglob.h"
 3  #include  "vgreg.h"
 4
 5  /*                                                                    *
 6   *   character read routine                                           *
 7   *   Returns the oldest keyboard character in the character queue.    *
 8   *   If no character has been input return -1.                        *
 9   *                                                                    *
10   *   NOTE:   The character is filled by a keyboard                    *
11   *   character interrupt causing vgkpiv() to be called.               *
12   *                                                                    */
13
14  vggetcar()
15      {
16      if(vgkflag == 0) return(-1);
17      vgkflag--;                                   //  removed one character
18      if(vgkquefl == NKQUE) vgkquefl = 0;
19      return(vgkque[vgkquefl++]>>8);
20      }
21
22
23
24
25
26  /*                                                                ●   *
27   *   Set/clear the blink mode on the picture, object, or element.     *
28   *                                                                    */
29
30  vgblink(type,num,action)
31      char type;                  //   0 - picture
32                                  //   1 - object
33                                  //   2 - element
34
35      int num;                    //   0 - picture
36                                  //   object or element num
37
38      char action;                //   0 - clear
39                                  //   1 - set
40      {
41
42      switch(type)
43          {
44          case PIC:               //   SET/CLEAR blink picture
45              {
46              return(vgpicmod(01000,action));
47              }
48          case OBJ:               //   SET/CLEAR blink on object 'num'
49              {
50              return(vgobjmod(num,01000,action));
51              }
52          case ELE:               //   SET/CLEAR blink on element 'num'
53              {
```

```
54                    return(vgelemod(num,01000,action));
55                }
56            }
57        }
58
59
60
61
62
63  /*                                                                          *
64   *    Modify the rotation matrix of the object.   The return values          *
65   *    are those of the object modification routine.                          *
66   *                                                                          */
67
68  vgrotate(num,x,y,z)
69       int num;                        //    object number
70       double x;                       //    radian rotation about X-axis
71       double y;                       //    radian rotation about Y-axis
72       double z;                       //    radian rotation about Z-axis
73       {
74       double sin();
75       double cos();
76       double sinx,siny,sinz,cosx,cosy,cosz;
77       sinx = sin(x);
78       siny = sin(y);
79       sinz = sin(z);
80       cosx = cos(x);
81       cosy = cos(y);
82       cosz = cos(z);
83       vgf_rot[0] = (cosz*cosy - sinx*siny*sinz)*2047;
84       vgf_rot[1] = (cosy*sinz + sinx*siny*cosz)*2047;
85       vgf_rot[2] = -siny*cosx*2047;
86       vgf_rot[3] = -sinz*cosx*2047;
87       vgf_rot[4] = cosx*cosz*2047;
88       vgf_rot[5] = sinx*2047;
89       vgf_rot[6] = (siny*cosz + sinx*cosy*sinz)*2047;
90       vgf_rot[7] = (siny*sinz - sinx*cosy*cosz)*2047;
91       vgf_rot[8] = cosx*cosy*2047;
92       return(vgobjmod(num,0i));
93       }
94
95
96
97
98
99  /*                                                                          *
100  *    Modify the X, Y, and Z coordinates of the object.   The range          *
101  *    of values are from -2048 (0177777) through 2047 (07777).               *
102  *    The return codes are those of the object modification routine.         *
103  *                                                                          */
104
105  vgcoord(num,x,y,z)
106       int num;                        //    object number
107       int x;                          //    X coordinate
108       int y;                          //    Y coordinate
109       int z;                          //    Z coordinate
110       {
111       vgf_dxr = x;
112       vgf_dyr = y;
113       vgf_dzr = z;
114       return(vgobjmod(num,04));
115       }
116
117
118
119
120
121  /*                                                                          *
122  *    Modify the intensity offset of the object.   The range of values      *
123  *    is from 0 to 1.   The return codes are those of the object             *
124  *    modification routine.                                      *
125  *                                                                          */
126
127  vgiofset(num,val)
128       int num;                            //    object number
```

73

```
129        double val;                    //   intensity offset value
130        {
131        vgf_ior = val*2047;            .
132        return(vgobjmod(num,010));
133        }
134
135
136
137
138
139 /*                                                                    *
140  *   Modify the intensity scale of the object.  The range of values   *
141  *   is from 0 to 1.  The return codes are those of the object        *
142  *   modification routine.                                            *
143  *                                                                    */
144
145 vgiscal(num,val)
146     int num;                          //   object number      .
147     double val;                       //   intensity scale
148     {
149     vgf_isr = val*2047;
150     return(vgobjmod(num,020));
151     }
152
153
154
155
156
157 /*                                                                    *
158  *   Modify the coordinate scale of the object.  The range of values  *
159  *   are 0 to 1.  The return codes are those of the object            *
160  *   modification routine.                                            *
161  *                                                                    */
162
163 vgcsr(num,val)
164     int num;                    .       //   object number
165     double val;                         //   coordinate scale value
166     {
167     vgf_csr = val*2047;
168     return(vgobjmod(num,02));
169     }
170
171
172
173
174                                              .          .
175 /*                                                                    *
176  *   Set/clear the light pen hookability of the object or element     *
177  *   The light pen halt, light pen interrupt, and light pen sense     *
178  *   switch are all treated as a single unit to be set/cleared.       *
179  *   Return codes are those of the modification routine concerned.    *
180  *                                                                    */
181
182 vglpen(type,num,action)
183     char type;                        //   1 - object.
184                                       //   2 - element
185
186     int num;                          //   object or element number
187
188     char action;                      //   0 - clear
189                                       //   1 - set
190     {
191     switch(type)
192         {
193         case OBJ:                              // set/clear object
194             {
195             return(vgobjmod(num,020400,action));
196             }
197         case ELE:                              // set/clear element
198             {
199             return(vgelemod(num,020400,action));
200             }
201         }
202     return(0);
203     }
```

74

```
204
205
206                                        75
207
208
209 /*                                                                                    *
210  *    Get the light pen interrupt registers.  The light pen                           *
211  *    interrupt flag is cleared so another light pen interrupt may                     *
212  *    be accepted by the light pen interrupt handler.                                  *
213  *                                                                                      */
214
215 vggetlpn(abp)
216      int abp;                                // 8 word buffer pointer
217      {
218      int *bp;
219      int i;
220      bp = abp;
221      for(i=0;i<7;i++)
222          *(bp++) = vglpbuf[i];
223      vglpflag = 0;
224      }
```

BIBLIOGRAPHY

1. Howard, J. F. and Thorpe, L. A., <u>Proposed Design
    Specification Manual for the Vector General Graphics
    Display Unit</u>, paper presented at Naval Postgraduate
    School CS4204 Class, June 1975

2. Howard, J. F. and Thorpe, L. A., <u>Proposed Users Manual
    for the Vector General Graphics Display Unit</u>, paper
    presented at the Naval Postgraduate School CS4202
    Class, June 1975

3. Naval Postgraduate School Technical Report
    NPS72Rr76031, <u>Users Manual for the Vector General
    Graphics Display Unit</u>, by L. A. Thorpe and G. W.
    Raetz, March 1976

4. Ritchie, D. E. and Thompson, K., The UNIX Timesharing
    System, <u>Communications of the ACM</u>, v. 17, no. 7, p
    365-375, July, 1974.

5. Vector General Inc., <u>Alphanumeric Keyboard KB1 Option
    Reference Manual</u>, Vector General, Inc., Woodland
    Hills, California May 1974

6. Vector General Inc., <u>Analog Devices Option Reference
    Manual</u>, Vector General Inc., Woodland Hills, Califor-
    nia, August 1974

7. Vector General Inc., <u>Circle-Arc Generator Reference
    Manual</u>, Vector General Inc., Canoga Park, California,
    February 1974

8. Vector General Inc., <u>Function Switch Option</u>, Vector
    General, Inc., Canoga Park, California, April 1974

9. Vector General Inc., Graphics Display System Reference
    Manual, Vector General Inc., Woodland Hill, Califor-
    nia, August 1974

10. Vector General Inc., Graphics Display System Technical
    Manual, Vector General Inc., Woodland Hills, Califor-
    nia, June 1974

11. Vector General Inc., Light Pen LP3 Option Reference
    Manual, Vector General Inc., Canoga Park, California,
    May 1974

12. Vector General Inc., PDP-11 Interface Option (with
    Sub-Stack) Reference Manual, Vector General Inc.,
    Woodland Hills, California, August 1974

Copies

Dean of Research   1
Code 023
Naval Postgraduate School
Monterey, California 93940

Defense Documentation Center   2
Cameron Station
Alexandria, Virginia 22314

Library (Code 0212)   2
Naval Postgraduate School
Monterey, California 93940

N. R. Church Computer Center   1
Naval Postgraduate School
Monterey, California 93940

Naval Electronics Systems Command (ELEX 320)   1
Attn: Cdr. A. Miller
Department of the Navy
Washington, D. C. 20360

Naval Electronics Laboratory   1
Center Library
271 Catalina Boulevard
San Diego, California 92152

Ltjg. Gary M. Raetz, USN (Code 72Rr)   3
Naval Postgraduate School
Monterey, California 93940

Professor George A. Rahe, (Code 72Ra)   1
Computer Science Group
Naval Postgraduate School
Monterey, California 93940

Lt. Lloyd A. Thorpe, USN   1
915 24th Street West
Billings, Montana 59102